1장. 데이터 마이닝 소개

예제 1. 핵심 인물 찾아라

사용자 목록 (그래프 노드)

In [1]:

```
users = [
    { "id": 0, "name": "Hero" },
    { "id": 1, "name": "Dunn" },
    { "id": 2, "name": "Sue" },
    { "id": 3, "name": "Chi" },
    { "id": 4, "name": "Thor" },
    { "id": 5, "name": "Clive" },
    { "id": 6, "name": "Hicks" },
    { "id": 7, "name": "Devin" },
    { "id": 8, "name": "Kate" },
    { "id": 9, "name": "Klein" }
]
```

친구 관계 목록 (그래프 에지)

In [2]:

```
friendship_pairs = [(0, 1), (0, 2), (1, 2), (1, 3), (2, 3), (3, 4),
                    (4, 5), (5, 6), (5, 7), (6, 8), (7, 8), (8, 9)]
```

Q. 친구 관계 목록 (인접 리스트)

In [3]:

```
# Initialize the dict with an empty list for each user id:
friendships = {user["id"]: [] for user in users}

# And loop over the friendship pairs to populate it:
# your code
for i, j in friendship_pairs:
    friendships[i].append(j)
    friendships[j].append(i)
```

Q. 친구 수 세기

In [4]:

```
def number_of_friends(user):
    """How many friends does _user_ have?"""
    user_id = user["id"]
    friend_ids = friendships[user_id]
    return len(friend_ids)
```

Q. 전체 친구 수 세기

In [5]:

```
total_connections = sum(number_of_friends(user) for user in users)


assert total_connections == 24
```

평균 친구 수 세기

In [6]:

```
num_users = len(users)                            # length of the users list
avg_connections = total_connections / num_users   # 24 / 10 == 2.4


assert num_users == 10
assert avg_connections == 2.4
```

Q. 사용자 별 친구 수 목록 생성

[(1, 3), (2, 3), (3, 3), (5, 3), (8, 3), (0, 2), (4, 2), (6, 2), (7, 2), (9, 1)]

In [7]:

```
# Create a list (user_id, number_of_friends).
num_friends_by_id = [(user["id"], number_of_friends(user)) for user in users]
```

In [8]:

```
num_friends_by_id.sort(                                # Sort the list
       key=lambda id_and_friends: id_and_friends[1],   # by num_friends
       reverse=True)                                   # largest to smallest

print(num_friends_by_id)
# Each pair is (user_id, num_friends):
# [(1, 3), (2, 3), (3, 3), (5, 3), (8, 3),
#  (0, 2), (4, 2), (6, 2), (7, 2), (9, 1)]

assert num_friends_by_id[0][1] == 3     # several people have 3 friends
assert num_friends_by_id[-1] == (9, 1)  # user 9 has only 1 friend
```

```
[(1, 3), (2, 3), (3, 3), (5, 3), (8, 3), (0, 2), (4, 2), (6, 2), (7, 2), (9, 1)]
```

예제2. 친구 추천하기

친구의 친구 목록 만들기 (않좋은 버전)

In [9]:

```
def foaf_ids_bad(user):
    """foaf is short for "friend of a friend" """
    return [foaf_id
            for friend_id in friendships[user["id"]]
            for foaf_id in friendships[friend_id]]

[0, 2, 3, 0, 1, 3]

assert foaf_ids_bad(users[0]) == [0, 2, 3, 0, 1, 3]
```

In [10]:

```
print(friendships[0])  # [1, 2]
print(friendships[1])  # [0, 2, 3]
print(friendships[2])  # [0, 1, 3]

assert friendships[0] == [1, 2]
assert friendships[1] == [0, 2, 3]
assert friendships[2] == [0, 1, 3]
```

```
[1, 2]
[0, 2, 3]
[0, 1, 3]
```

Q. 친구의 친구 목록 만들기

In [11]:

```
from collections import Counter                    # not loaded by default

# 나와 친구를 제외한 친구의 친구 목록 만들기
def friends_of_friends(user):
    user_id = user["id"]
    return Counter(
        foaf_id
        for friend_id in friendships[user_id]
        for foaf_id in friendships[friend_id]
        if foaf_id != user_id
        and foaf_id not in friendships[user_id]
    )

print(friends_of_friends(users[3]))               # Counter({0: 2, 5: 1})
assert friends_of_friends(users[3]) == Counter({0: 2, 5: 1})
```

```
Counter({0: 2, 5: 1})
```

같은 관심을 갖는 친구 추천하기

In [12]:

```
interests = [
    (0, "Hadoop"), (0, "Big Data"), (0, "HBase"), (0, "Java"),
    (0, "Spark"), (0, "Storm"), (0, "Cassandra"),
    (1, "NoSQL"), (1, "MongoDB"), (1, "Cassandra"), (1, "HBase"),
    (1, "Postgres"), (2, "Python"), (2, "scikit-learn"), (2, "scipy"),
    (2, "numpy"), (2, "statsmodels"), (2, "pandas"), (3, "R"), (3, "Python"),
    (3, "statistics"), (3, "regression"), (3, "probability"),
    (4, "machine learning"), (4, "regression"), (4, "decision trees"),
    (4, "libsvm"), (5, "Python"), (5, "R"), (5, "Java"), (5, "C++"),
    (5, "Haskell"), (5, "programming languages"), (6, "statistics"),
    (6, "probability"), (6, "mathematics"), (6, "theory"),
    (7, "machine learning"), (7, "scikit-learn"), (7, "Mahout"),
    (7, "neural networks"), (8, "neural networks"), (8, "deep learning"),
    (8, "Big Data"), (8, "artificial intelligence"), (9, "Hadoop"),
    (9, "Java"), (9, "MapReduce"), (9, "Big Data")
]
```

Q. 관심 별 사용자 목록 구성

In [13]:

```
from collections import defaultdict

# Keys are interests, values are lists of user_ids with that interest
user_ids_by_interest = defaultdict(list)

for user_id, interest in interests:
    user_ids_by_interest[interest].append(user_id)
```

Q. 사용자 별 관심 목록 구성

In [14]:

```
# Keys are user_ids, values are lists of interests for that user_id.
interests_by_user_id = defaultdict(list)

for user_id, interest in interests:
    interests_by_user_id[user_id].append(interest)
```

Q. 같은 관심을 갖는 사람들 목록 (Counter 형태로 반환)

In [15]:

```
def most_common_interests_with(user):
    return Counter(
        interested_user_id
        for interest in interests_by_user_id[user["id"]]
        for interested_user_id in user_ids_by_interest[interest]
        if interested_user_id != user["id"]
    )
```

In [16]:

```
print(most_common_interests_with(users[0]).most_common())
```

```
[(9, 3), (1, 2), (8, 1), (5, 1)]
```

예제3. 연봉과 근속연수의 관계를 찾아라

직원들의 연봉 및 근속연수 테이블

In [17]:

```
salaries_and_tenures = [(83000, 8.7), (88000, 8.1),
                        (48000, 0.7), (76000, 6),
                        (69000, 6.5), (76000, 7.5),
                        (60000, 2.5), (83000, 10),
                        (48000, 1.9), (63000, 4.2)]
```

근속연수 별 평균 연봉 계산

In [18]:

```
# Keys are years, values are lists of the salaries for each tenure.
salary_by_tenure = defaultdict(list)

for salary, tenure in salaries_and_tenures:
    salary_by_tenure[tenure].append(salary)

# Keys are years, each value is average salary for that tenure.
average_salary_by_tenure = {
    tenure: sum(salaries) / len(salaries)
    for tenure, salaries in salary_by_tenure.items()
}
```

In [19]:

```
assert average_salary_by_tenure == {
    0.7: 48000.0,
    1.9: 48000.0,
    2.5: 60000.0,
    4.2: 63000.0,
    6: 76000.0,
    6.5: 69000.0,
    7.5: 76000.0,
    8.1: 88000.0,
    8.7: 83000.0,
    10: 83000.0
}
```

근속연수 버킷 구성

In [20]:

```
def tenure_bucket(tenure):
    if tenure < 2:
        return "less than two"
    elif tenure < 5:
        return "between two and five"
    else:
        return "more than five"
```

근속연수 버킷 별 연봉 리스트 구성

In [21]:

```
# Keys are tenure buckets, values are lists of salaries for that bucket.
salary_by_tenure_bucket = defaultdict(list)

for salary, tenure in salaries_and_tenures:
    bucket = tenure_bucket(tenure)
    salary_by_tenure_bucket[bucket].append(salary)
```

근속연수 버킷 별 평균 연봉 계산

In [22]:

```
# Keys are tenure buckets, values are average salary for that bucket
average_salary_by_bucket = {
  tenure_bucket: sum(salaries) / len(salaries)
  for tenure_bucket, salaries in salary_by_tenure_bucket.items()
}
```

In [23]:

```
assert average_salary_by_bucket == {
    'between two and five': 61500.0,
    'less than two': 48000.0,
    'more than five': 79166.66666666667
}
```

예제4. 유료 계정 전환 대상자를 찾아라

In [24]:

```
def predict_paid_or_unpaid(years_experience):
  if years_experience < 3.0:
    return "paid"
  elif years_experience < 8.5:
    return "unpaid"
  else:
    return "paid"
```